

# Google App Script

## O que é?

O Google App Script (GAS) é uma plataforma de **desenvolvimento de aplicativos**, associados aos produtos da Google ou de terceiros. Assim, o desenvolvedor pode **utilizar diversos recursos do Google Workspace** e integrar diversas APIs em seu programa.



Sua **sintaxe é similar ao Java**, sendo assim, sua linguagem é baseada em [Programação Orientada a Objetos](#).

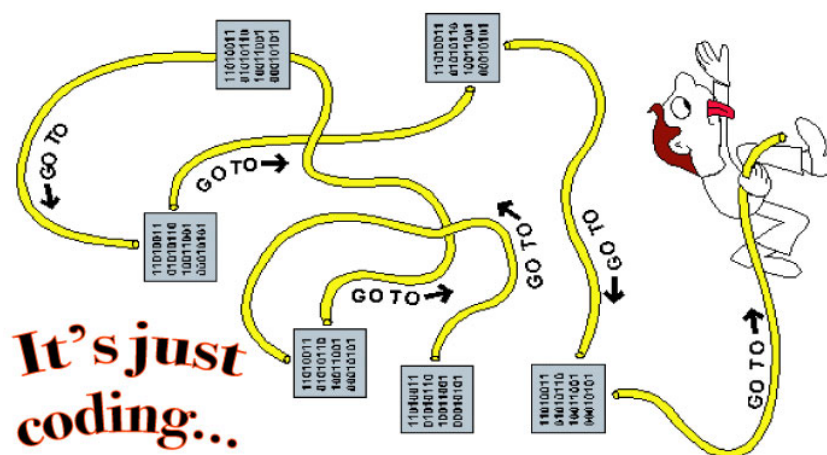
---

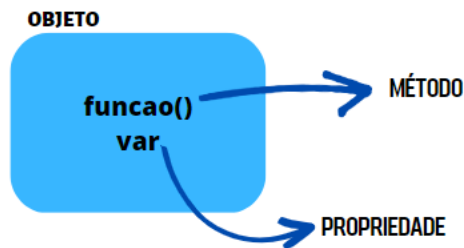
## Programação Orientada a Objetos (POO)?

Quando aprendemos programação, somos orientados a programar de modo procedural. Isso significa que, para construirmos um algoritmo, organizamos as instruções para serem executadas de modo procedural e sequencial, isto é, cada procedimento de cada vez.

Porém, ao construirmos projetos maiores e mais complexos, é cada vez mais difícil manter um código bem estruturado programando desse modo. Dessa forma, por não termos alternativas fáceis de reutilizar código, é comum termos que copiar e colar repetidamente linhas do programa. Além disso, existe o problema de funções com uma interdependência excessiva (uma função ligada à outra, ligada a outra, ligada a anterior...), o que pode gerar uma estrutura muito complexa no código.

O nome que se dá para esse código mal estruturado e de difícil compreensão é “**código spaghetti**”. Por conta disso, surge o paradigma de POO para criar softwares mais flexíveis, eficientes, bem estruturados e reutilizáveis.





O conceito principal da POO é o **Encapsulamento**, que é quando agrupamos variáveis relacionadas e funções que operam nelas em **objetos**, que, por sua vez, representam uma entidade ou conceito do mundo real. Podemos chamar essas variáveis de **propriedades/atributos** do objeto, e, de **métodos**, as funções que realizam tarefas ou interagem com outros objetos.

Exemplificando o conceito, suponha que você deseja criar um sistema de reserva de hotéis. Como começar? É simples: basta incorporar os princípios da Programação Orientada a Objetos (POO) ao seu código.

Para representar a entidade do mundo real "Hotel" em programação orientada a objetos, podemos criar um objeto com um atributo "nome". Além disso, podemos criar um método chamado "reservarQuarto()" que interage com esse objeto e permite que um usuário reserve um quarto no hotel.



Por fim, você pode representar esse encapsulamento em código com a **notação dot**. Para acessar os atributos do objeto, basta seguir a notação como no código a seguir:

```
\\ Objeto.atributo
Hotel.nome > "Hotel Vritago"
```

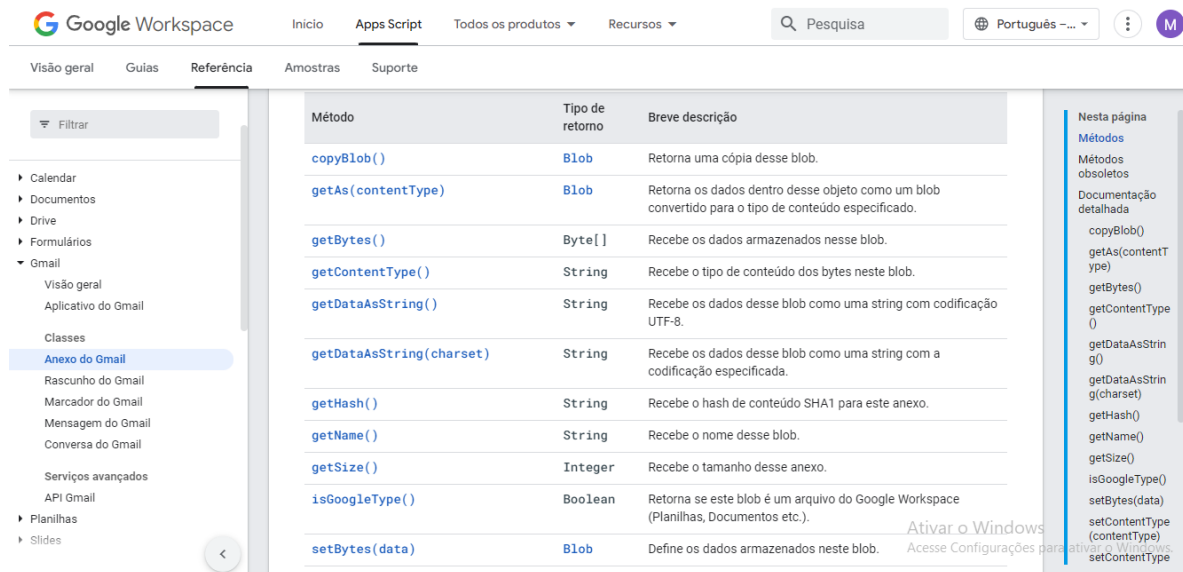
O princípio segue para acessar os métodos do objeto:

```
\\ Objeto.metodo()
Hotel.reservarQuarto() > "Quarto Reservado"
```

Por fim, vale destacar o conceito de **classe**. Uma classe pode ser vista como um modelo que define as características e comportamentos que os objetos terão. Por exemplo, o objeto "Hotel" poderia ser uma instância da classe "Edificações", que por sua vez poderia incluir outros objetos como "Casa", "Museu" e "Hospital".

Já as principais formas nas quais as classes interagem entre si podem ser dadas pelo conceito de **Herança e Polimorfismo**. Embora esses conceitos não sejam abordados neste relatório por razões pragmáticas, é importante ressaltar que entendê-los é essencial para o entendimento completo de POO.

O GAS fornece diversos métodos para os objetos que você queira usar, e você pode [acessar a documentação](#) para conferir as possibilidades de desenvolvimento do seu programa:



Método	Tipo de retorno	Breve descrição
<code>copyBlob()</code>	<code>Blob</code>	Retorna uma cópia desse blob.
<code>getAs(contentType)</code>	<code>Blob</code>	Retorna os dados dentro desse objeto como um blob convertido para o tipo de conteúdo especificado.
<code>getBytes()</code>	<code>Byte[]</code>	Recebe os dados armazenados nesse blob.
<code>getContentType()</code>	<code>String</code>	Recebe o tipo de conteúdo dos bytes neste blob.
<code>getDataAsString()</code>	<code>String</code>	Recebe os dados desse blob como uma string com codificação UTF-8.
<code>getDataAsString(charset)</code>	<code>String</code>	Recebe os dados desse blob como uma string com a codificação especificada.
<code>getHash()</code>	<code>String</code>	Recebe o hash de conteúdo SHA1 para este anexo.
<code>getName()</code>	<code>String</code>	Recebe o nome desse blob.
<code>getSize()</code>	<code>Integer</code>	Recebe o tamanho desse anexo.
<code>isGoogleType()</code>	<code>Boolean</code>	Retorna se este blob é um arquivo do Google Workspace (Planilhas, Documentos etc.).
<code>setBytes(data)</code>	<code>Blob</code>	Define os dados armazenados neste blob.

### **Conferindo os métodos possíveis na documentação relacionados a anexo do Gmail**

Seu editor é executado pelo lado de servidor, ou seja, **não é necessário instalar nada para usá-lo**. Para começar a usar somente é preciso uma conexão a internet e uma conta Gmail.

Com o GAS é possível:

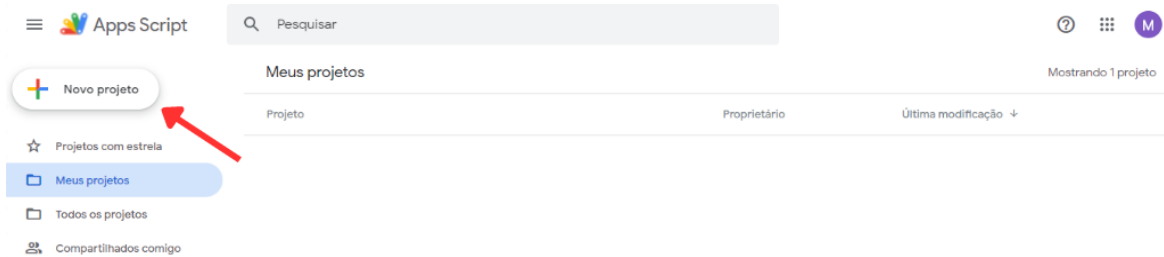
- Criar funcionalidade especiais para Google Sheets, GoogleDocs, Gmail e outros produtos relacionados
- Interagir funcionalidade entre os produtos da Google mencionados
- Converter seu AppScript em um webApp
- Usar Cloud Computing para desenvolver programas sem baixar nada

[Acessar a home do Google App Script](#)

## **Guia Rápido**

### **Os Básicos:**

Iremos, agora, dar início a uma introdução da sintaxe característica do Google App Script. Primeiro, [abra seu editor](#), e crie um novo projeto para começar a programar:



Para **criar uma função**, basta escrever:

```
function minhaFuncao() {  
    /* Instruções da sua função */  
}
```

Você pode **declarar variáveis** utilizando `var` ou `let`:

```
function minhaFuncao(){  
    // Com "var", a variável é declarada para o escopo de toda função  
    var x;  
  
    {  
        // Com "let", a variável é declarada para o escopo do bloco ({})  
        let z;  
    }  
}
```

Utilizando declarações **if/else**:

```
function minhaFuncao(){  
    var a;  
    a=10;  
  
    if(a<20)  
    {  
        // O objeto Logger é usado para registrar mensagens para um sistema.  
        // Assim, podemos ver o resultado esperado do output no console do GAS!  
        Logger.log("Opção 1");    }  
    else  
    {  
        Logger.log("Opção 2");  
    }  
}
```

Imprimindo a quantidade de iterações realizadas, usando declarações de **for loop**:

```
function minhaFuncao(){

    for(var i=1;i<6;i++){
        Logger.log("iteracao "+i)
    }
}
```

Imprimindo números de 1 a 5, usando as declarações **while** e **do while**:

```
function minhaFuncao(){
    var i=1;
    while(i<6)
    {
        Logger.log(i);
        i++;
    }
    do
    {
        i++;
        Logger.log(i)
    }while(i<6);
}
```

Criando um menu, implementando um **Switch Case**:

```
function minhaFuncao(){
    var opcao=2;

    switch(opcao){
        case 1:
            Logger.log("Opcao 1")
            break;
        case 2:
            Logger.log("Opcao 2")
            break;
        default:
            Logger.log("Outra opcao")
            break;
    }
}
```

Mostrando uma lista de frutas, utilizando **vetores** e suas operações:

```
function minhaFuncao(){

    var arr1=["Banana", "Maçã","Pera"];
    //mudando elemento do index 0, o primeiro elemento
    arr1[0]="Melancia"

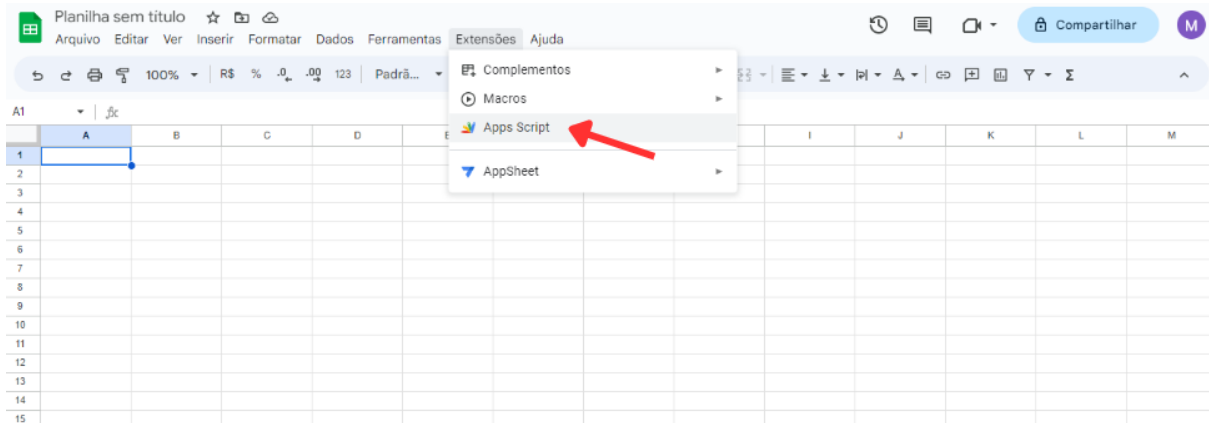
    // Adicionando Banana ao final do vetor
    arr1.push("Banana")
}
```

```
// Imprimindo as duas listas em uma única concatenada
arr2 = ["Abacate", "Limão", "Uva"]
Logger.log(arr1.concat(arr2))
}
```

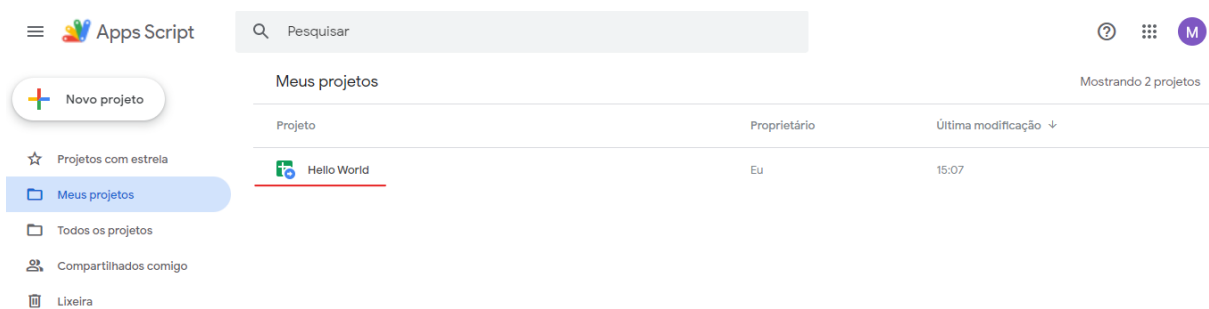
### **Meu primeiro Projeto:**

A seguir, iremos adicionar uma nova funcionalidade básica na planilha Google: criar um pop up mostrando uma mensagem ao usuário.

Primeiro, vamos criar um novo projeto. Você poderá fazer isso diretamente pelo Google Sheets, acessando Extensões e clicando Apps Script:



Você poderá gerenciar esse e outros projetos posteriormente na aba [Meus Projetos](#):



A seguir, você pode escrever sua nova funcionalidade. Abaixo, veremos um código simples:

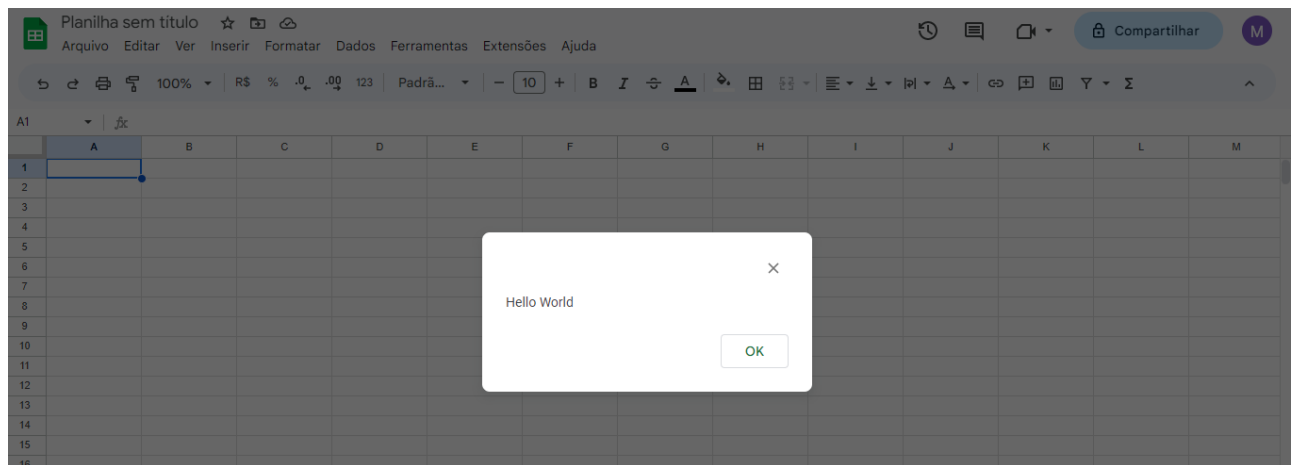
```
function escrever() {
  // Armazenando o conteúdo da mensagem em uma variável
  var minha_mensagem="Hello World";

  //Selecione o objeto SpreadsheetApp, e chamando um método para
  modificar seu conteúdo
```

```
//getUI irá acessar a interface do usuário, e o outro método alert
mostrará um pop up com a mensagem fornecida
SpreadsheetApp.getUi().alert(minha_mensagem);
}
```

Esse código cria uma função chamada "escrever" que armazena uma mensagem "Hello World" na variável "minha\_mensagem". Em seguida, utiliza o objeto "SpreadsheetApp" para acessar a interface do usuário do Google Sheets, e mostra um pop-up contendo "minha\_mensagem", usando o método "alert()".

Finalmente, ao executarmos o código e retornarmos a nossa planilha original, iremos ver essa nova funcionalidade:



## Limitações

→ O tempo de execução máximo suportado por script é de 6 minutos, e [30 minutos](#) em quaisquer contas empresariais pagas ou contas educacionais. Para contornar isso você pode:

- ◆ Pesquisar maneiras de tornar seu código mais eficiente,
- ◆ Buscar métodos que permitam você executar seu script várias vezes, até que todo seu script seja executado.

[Exemplo](#)

→ Existe uma série de [Cotas e Limites de uso para os serviços Google](#), são algumas delas:

	Conta <a href="#">Legado</a> (Gratuito)	Conta Workspace(Pago)
Slides criados	250 / dia	1.500 / dia

Planilhas criadas	250 / dia	3.200 / dia
Execuções simultâneas	30 / usuário	30 / usuário
Anexos de e-mails	250 / msg	250 / msg
Tamanho do corpo do e-mail	200 KB / msg	400 KB / msg
Destinatários de e-mail por mensagem	50 / msg	50 / msg
Tamanho total dos anexos do e-mail	25 MB / msg	25 MB / msg
Documentos criados	250 / dia	1.500 / dia
Arquivos convertidos	2.000 / dia	4.000 / dia
Destinatários de e-mail por dia	100* / dia	1.500* / dia
Destinatários de e-mail por dia no domínio	100* / dia	2.000 / dia
Leitura/gravação de e-mail (excluindo enviar)	20.000 / dia	50.000 / dia